Challenges and Opportunities of FPGA Acceleration in Big Data

Neetu Reji[1], Smitha C Thomas[2], Reshma Suku.[3]

[1]M. Tech Student, Computer Science and Engineering, APJ Abdul Kalam Technological University, Kerala,India
[23]Asst. Professor, Computer Science and Engineering, Mount Zion College of Engineering, Kadammanitta, Kerala, India

---***---

Abstract - *FPGA with customized IP helps to lower the power consumption to accelerate computation intensive segment for the application and optimize the performance. To transfer raw data from source server to data ware house ETL procedure is used in big data field. FPGA has been noticed in the industry because of its performance- re-programmable flexibility, per-power efficiency, and wide range of applicableness. In this paper we will discuss how programmable gate arrays help a Spark ETL workload to reduce high CPU utilization issue. It should release more CPU power to run some compute intensive jobs. Also we will discuss about benefits of FPGA in deep learning applications for AI.*

Key Words: **Cloud computing, Computational Modeling, Field Programmable Gate Arrays, Parallel Processing, Flash EPROM**.

1. INTRODUCTION

"Big Data" is a broad term for datasets that are so large or complex. Workflows are the task oriented and often require more specific data than process. The Process is designed on a higher level scenarios that helps for decision making in organizational level. Big Data workflow is best illustrated in comparing traditional IT workloads with Big Data workloads. It may require many servers to run one application whereas traditional IT workloads requires one server to run many application. Big Data workloads run to the completion and traditional IT workloads run forever. The scale of big data will be representing the volume, velocity and variety of data. Volume indicate how much amount of data is generated. Velocity is used to indicate the speed of generating data and data generated in real time. Variety indicates the wide range in which the data can be encode.

To increasing the processing capacity has been a main area of prior research. Examples include CPU optimizations and the use of dedicated hardware accelerators such as GPUs. Another accelerator is the field programmable gate array or FPGA. These FPGAs consist of a re-configurable fabric that can be programmed to implement custom integrated circuit (IC) designs. This work investigates how these FPGA accelerators can be efficiently deployed to increase processing capacity in a big data context.

Nowadays in every field of industry we are using some form of data analytics. The impact and possibilities of transparent and efficient integration of FPGAs in big data frameworks are therefore limitless. The three areas of applications that could benefit from FPGA accelerated big data frameworks; applications with long-running queries, latency sensitive applications, and applications that aim to achieve a high energy efficiency.

Challenges: There are two main challenges in integrating FPGA accelerators with big data frameworks are transparency and efficiency. The user should not be aware of the FPGA acceleration and does not have to tune certain parameters in the framework. It is important because transparent integration lowers the barrier to adopt these technologies. The system should autonomously identify where and when certain parts of the computation can be accelerated to achieve transparency. There are two factors that play a role in the context of FPGA accelerators. First, the initial cost of developing an FPGA. It is generally more time consuming than software development for CPUs and GPUs, and requires in-depth knowledge about circuit design. Lastly, these FPGA accelerators are expensive. The industry standard is to run big data applications in a cloud environment so there is no need for end-users to buy any specialized hardware.

FPGA is an IC form that have internal logic design which we can configure after manufacturing helps programmer to implement different IC without having to go through the manufacturing process, which is time consuming and expensive. This reconfiguration of the FPGA is done using description language. Data is processed in a dataflow manner. FPGAs implementing dataflow-oriented architectures with high levels of (pipeline) parallelism can provide high application throughput, often providing high energy efficiency. Latency-sensitive applications can leverage FPGA accelerators by directly connecting to the physical layer of a network, and perform data transformations without going through the software stacks of the host system. While these advantages of FPGA accelerators hold promise, difficulties associated with programming and integration limit their use.

It can be integrated into big data systems, can discriminate into three configuration of the FPGA in the system. The accelerator can either be placed in the data path between network or storage and the CPU. It can be made between an IO-attached accelerator, where the FPGA has its own memory space, and a co-processor, in which the FPGA and the CPU communicate through shared memory.

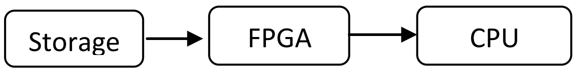

Fig 1: Data Path

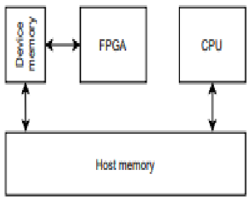

Fig 2: IO attached

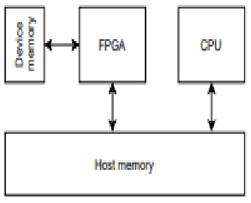

Fig 3: Co-processor

SQL workloads: It is target to leverage FPGA highly parallel computing capability to accelerate Spark SQL Query and for FPGA's higher power efficiency than CPU we can lower the power consumption at the same time. The Architecture consists of SQL query decomposition algorithms, fine-grained FPGA based Engine Units which perform basic computation of sub string, arithmetic and logic operations. Using SQL query decomposition algorithm, we are able to decompose a complex SQL query into basic operations and according to their patterns each is fed into an Engine Unit. SQL Engine Units are highly configurable and can be chained together to perform complex Spark SQL queries, finally one SQL query is transformed into a Hardware Pipeline.

2. Speedup Methods

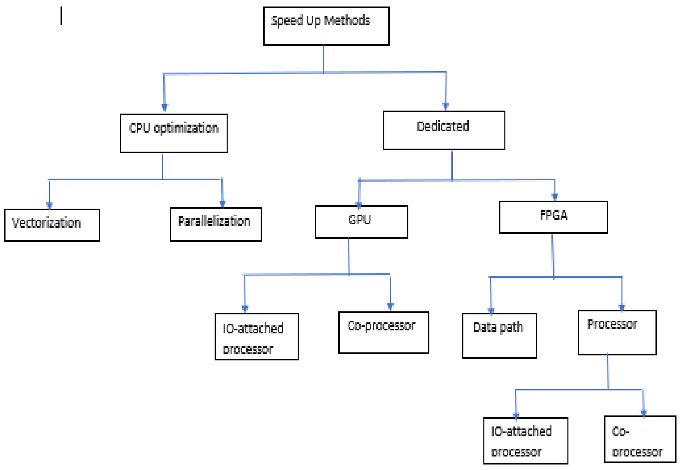

Fig 4: Classification tree for speedup methods

CPU optimization: It depends on available CPU. Modern CPUs offer vectorized operations. It perform a single instruction on a vector which contain multiple values. Size depend on underlying support hardware. Parallelization can be establish in the form of multithreading. It can be executed on multiple cores, increasing the effective throughput. It is important to select the correct granularity when implementing a multithreading solution, as an overhead is associated with the creation and the management of multiple threads.

GPU accelerators: GPUs have proven to be very efficient for training and running deep learning models.in fact, has even pivoted from a pure GPU and gaming company to a provider of cloud GPU services and a competent AI research lab. GPUS that are configured as an IO-attached processor operate in their own memory space, separated from the memory space of the CPU. Sharing data between the host and the GPU accelerator requires the data to be copied between the two memory spaces. GPUs that are configured as a co-processor share their memory space with the CPU. The accelerator can therefore access data in the main memory without the need to copy this data.

FPGA accelerators: The inherent highly flexible fine-grained parallelism of FPGA accelerators offers data, task, and pipeline parallelism, resulting in faster data process execution. FPGA accelerators have already set to expand beyond the data centers. FPGAs in tha data path are placed between network or storage and the CPU. These FPGAs can perform preprocessing operations, such as parquet decompression, which effectively increases the bandwidth between network or storage and the CPU. FPGA accelerator acts as another processor in parallel with the CPU. The FPGA has it's own memory space, requiring data to be copied between the CPU and the accelerator. Another drawback of such a system, apart from the required data copies, is the limited capacity of local memory that is available on the FPGA. Because of this limitation, applications that run on FPGA accelerators can potentially only access a fraction of the entire dataset at the same time.

3. Proposed solutions

Big data framework utilizes vectorization in combination with GPU and FPGA acceleration that is perfectly viable solution. The scope of this project is to the implementation of FPGA accelerators, but it is good to know what the strengths of other speedup methods are. F or implementation the processor class solution is chosen , due to it's slightly higher availability. In addition, it's strengths best fit the acceleration needs of SQL operators. This class is further subdivided into FPGA processors that act as a co-processor or IO-attached processor. The availability of running as a co-processor is heavily dependent on the support of the underlying system such as OpenCAPI. Three different solutions are proposed. All of the proposed solutions make use of FPGA accelerators configured as an IO-attached processor, since the

FPGA kernel as deployed on Amazon Web Services (AWS) does not support the co-processor configuration at the time of writing.

All data structures are stored using the Apache Arrow in-memory format. An implementation that utilizes both CPU optimization methods as well as FPGA accelerators is a good demonstration of multiple big data speedup methods working together. Another it is chosen as the second framework for acceleration with FPGA accelerators configured as processors. These are identified as the most promising candidates for integration based on the required development effort, expected performance, and potential impact. Finally, Dask distributed is selected for the third integration. It is specifically designed to run in a cluster setting and is therefore the perfect candidate to investigate a cluster deployment with an acceleration aware worker node.

4.Implementation

Sabot planner is based on the open-source Apache Calcite SQL optimizer.An FPGA kernel that can perform regular expression matching is selected for integration.It is extended in two places : planner & operator package. Sabot planner has number of planning phases.Each phase transpose the coming execution tree by use of a number of optimization rules.The validation phase validates the query based on the schema of the dataset. In the next phase, the SQL query is parsed and converted to an expression tree of logical operations. This tree is then optimized in a number of logical planning phases.

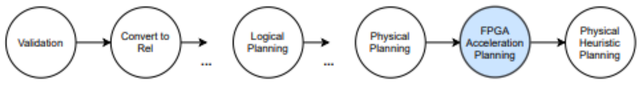

Fig 5: Sabot planner phases including the new FPGA acceleration planning phase.

Developing an efficient FPGA implementation of a SQL operator is labor intensive.FPGA acceleration planning phase such that they can match the optimization rules in cases where they normally would not. The FPGA acceleration planning phase transforms the physical execution plan by substituting the filter operator with the accelerated version.The accelerated operator can either offload the evaluation of the filter to the RE2 library or to an FPGA accelerator through the Tidre package.The the results of accelerating Dremio with the RE2 library, which implements a regular expression engine optimized for the CPU.

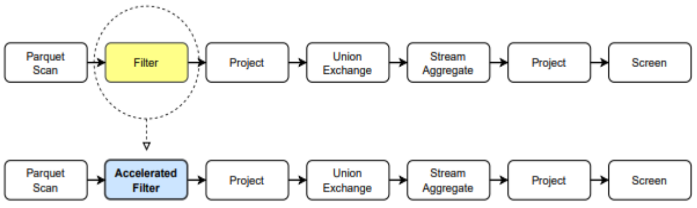

Fig 6: Execution plan transformation for the regular expression use case.

Dask distributed is a different version of Dask which can be run on a cluster. It provides the potential for added compute power and parallelization, but at the same time it also adds additional complexity. The client node is controlled by the end-user. It is used for all interactions within cluster. The scheduler node is used by the client to communicate, which keeps records of all worker nodes in the cluster. These worker nodes are held in an abstraction known as the worker pool. The accelerated version is deployed as an acceleration aware worker setup for this framework.

The accelerated worker implementation is the only new contribution required to accelerate Dask distributed. The FPGA acceleration planning stage and the accelerated operator implementations can be imported from the accelerated version.

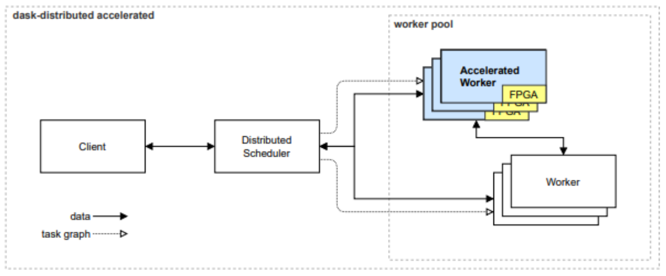

Fig 6: System architecture of Dask distributed including the accelerated worker node

Data set can be submit by client node to the cluster., it is scattered over the worker pool. To submit queries we are using this connection. The scheduler plan these queries and sends the resulting task graph to the worker nodes in the worker pool. It will not send as a whole, so individual task are submitted to the workers. It will execute these tasks. The worker node send data between each other in order to satisfy the missing dependencies of these tasks. The accelerated worker node implementation performs the FPGA acceleration stage on all incoming subgraphs of the task graph. Therefore, only accelerated worker nodes require an FPGA accelerator to be installed, as the vanilla worker implementations execute the original task subgraphs without any accelerated operators.

5. CONCLUSIONS

In this paper we identified that FPGA accelerators can either be placed in the data path or attached as a processor. , FPGA accelerators attached as a processor, is best suited for the acceleration of individual SQL operators in an existing big data framework. A distinction is made between FPGA accelerators configured as an IO-attached processor and FPGA accelerators configured as a co-processor. From these two configurations, FPGA accelerators configured as a co-processor are preferable, but this configuration is only available if the underlying compute system supports this. This work integrates FPGA accelerators configured as an IO-attached processor into three batch-processing big data frameworks. Additionally, it is found that reading data from disk takes up a significant portion of the runtime of a SQL workload. In this case, FPGA accelerators can be placed in between storage and the CPU. These accelerators could perform algorithms such as parquet decompression, projection, or filtering to reduce the runtime of these operations. Two absolute requirements for the efficient integration of FPGA accelerator in big data frameworks are the presence of a flexible API for the manipulation of execution plans and the use of a hardware-friendly and language-independent in-memory data format such as Apache Arrow.

REFERENCES

[1] Robert Chang. A beginner's guide to data engineering — part ii, . URL https://medium. com/@rchang/a-beginners-guide-to-data-engineering-part-ii-47c4e7cbda71. accessed on 2021-06-09.

[2] https://ieeexplore.ieee.org/document/9439431/metrics#metrics

[3] https://www.semanticscholar.org/paper/FPGA-Acceleration-for-Big-Data-Analytics%3A-and-Hoozemans-Peltenburg/76c087bd67124a3b3d8df3c8100564d14cf5f1b0

[4] https://www.stemmer-imaging.com/en/technical-tips/introduction-to-fpga-acceleration/#:~:text=An%20FPGA%20(Field%20Programmable%20Gate,together%20in%20many%20different%20ways

[5] https://www.intel.com/content/www/us/en/artificial-intelligence/programmable/fpga-gpu.html.

[6] https://bdtechtalks.com/2020/11/09/fpga-vs-gpu-deep-learning/

[7] https://sites.usc.edu/fpga/big-data-analytics-on-heterogeneous-architectures/

[8] https://www.datacenterdynamics.com/en/opinions/turning-big-data-challenges-opportunities-fpga-accelerated-computing/

BIOGRAPHIES

Neetu Reji, currently pursuing MTech degree in Computer Science and Engineering from APJ Abdul Kalam Technological University, Kerala, India at Mount ZionCollege of Engineering, Kadammanitta, Kerala, India.

Smitha C Thomas received the MTech degree in Computer science and Engineering.. She is currently working as Assistant Professor in the Department of Computer science and Engineering at Mount Zion College of Engineering, Kadammanitta, Kerala, India.

Reshma Suku received the MTech degree in Computer science and Engineering.. She is currently working as Assistant Professor in the Department of Computer science and Engineering at Mount Zion College of Engineering, Kadammanitta, Kerala, India.